A/B Testing the Udacity Website

IDS 701: Solving Problems with Data

Tea Tafaj Tonantzin Real Rojas

2026-01-27

In these exercises, we'll be analyzing data on user behavior from an experiment run by Udacity, the online education company. More specifically, we'll be looking at a test Udacity ran to improve the onboarding process on their site.

Udacity's test is an example of an "A/B" test, in which some portion of users visiting a website (or using an app) are randomly selected to see a new version of the site. An analyst can then compare the behavior of users who see a new website design to users seeing their normal website to estimate the effect of rolling out the proposed changes to all users. While this kind of experiment has it's own name in industry (A/B testing), to be clear it's just a randomized experiment, and so everything we've learned about potential outcomes and randomized experiments apply here.

(Udacity has generously provides the data from this test under an Apache open-source license, and you can find their original writeup here. If you're interested in learning more on A/B testing in particular, it seems only fair while we use their data to flag they have a full course on the subject here.)

Udacity's Test

The test is described by Udacity as follows:

At the time of this experiment, Udacity courses currently have two options on the course overview page: "start free trial", and "access course materials".

Current Conditions Before Change

- If the student clicks "start free trial", they will be asked to enter their credit card information, and then they will be enrolled in a free trial for the paid version of the course. After 14 days, they will automatically be charged unless they cancel first.

- If the student clicks "access course materials", they will be able to view the videos and take the quizzes for free, but they will not receive coaching support or a verified certificate, and they will not submit their final project for feedback.

Description of Experimented Change

- In the experiment, Udacity tested a change where if the student clicked "start free trial", they were asked how much time they had available to devote to the course.
- If the student indicated 5 or more hours per week, they would be taken through the checkout process as usual. If they indicated fewer than 5 hours per week, a message would appear indicating that Udacity courses usually require a greater time commitment for successful completion, and suggesting that the student might like to access the course materials for free.
- At this point, the student would have the option to continue enrolling in the free trial, or access the course materials for free instead. This screenshot shows what the experiment looks like.

Udacity's Hope is that...:

> this might set clearer expectations for students upfront, thus reducing the number of frustrated students who left the free trial because they didn't have enough time – without significantly reducing the number of students to continue past the free trial and eventually complete the course. If this hypothesis held true, Udacity could improve the overall student experience and improve coaches' capacity to support students who are likely to complete the course.

Gradescope Autograding

Please follow all standard guidance for submitting this assignment to the Gradescope autograder, including storing your solutions in a dictionary called `results` and ensuring your notebook runs from the start to completion without any errors.

For this assignment, please name your file `exercise_abtesting.ipynb` before uploading.

You can check that you have answers for all questions in your `results` dictionary with this code:

```
assert set(results.keys()) == {
    "ex4_avg_oec",
    "ex5_avg_guardrail",
    "ex7_ttest_pvalue",
    "ex9_ttest_pvalue_clicks",
    "ex10_num_obs",
    "ex11_guard_ate",
    "ex11_guard_pvalue",
    "ex11_oec_ate",
    "ex11_oec_pvalue",
    "ex14_se_treatment",
}
```

Submission Limits

Please remember that you are **only allowed THREE submissions to the autograder.** Your last submission (if you submit 3 or fewer times), or your third submission (if you submit more than 3 times) will determine your grade Submissions that error out will **not** count against this total.

Import the Data

```
import pandas as pd
import numpy as np
from scipy import stats

# quarto preview exercise_abtesting.ipynb --to pdf

results = {}
```

Exercise 1

Begin by importing Udacity's data on user behavior here.

There are TWO datasets for this test — one for the control data (users who saw the original design), and one for treatment data (users who saw the experimental design). Udacity decided to show their test site to 1/2 of visitors, so there are roughly the same number of users appearing in each dataset (though this is not a requirement of AB tests).

Please rmeember to load the data directly from github to assist the autograder.

```
control_path = "https://github.com/nickeubank/MIDS_Data/raw/refs/heads/master/udacity_AB_tes
tmt_path = "https://github.com/nickeubank/MIDS_Data/raw/refs/heads/master/udacity_AB_testing

df_control = pd.read_csv(control_path)
df_tmt = pd.read_csv(tmt_path)
```

```python
print(f"Control: {df_control.shape}")
print(f"Treatment: {df_tmt.shape}")
```

```
Control: (37, 5)
Treatment: (37, 5)
```

Exercise 2

Explore the data. Can you identify the unit of observation of the data (e.g. what is represented by each row)?

To be clear, the columns represent stages in a user funnel:

- Some number of users arrive at the website and are counted as Pageviews,
- Some portion of those users then click to enroll (and are counted as clicks),
- Some portion of those users then actually enroll in the free trial (after seeing an informational popup, in the case of treatment individuals),
- Finally some portion of those users end up paying at the end of the free trial period.

(Note this is not the only way that A/B test data can be collected and/or reported — this is just what Udacity provided, presumably to help address privacy concerns.)

```python
print("Control")
df_control.head(3)
```

```
Control
```

	Date	Pageviews	Clicks	Enrollments	Payments
0	Sat, Oct 11	7723	687	134.0	70.0
1	Sun, Oct 12	9102	779	147.0	70.0
2	Mon, Oct 13	10511	909	167.0	95.0

```python
print("Treatment")
df_tmt.head(3)
```

```
Treatment
```

	Date	Pageviews	Clicks	Enrollments	Payments
0	Sat, Oct 11	7716	686	105.0	34.0
1	Sun, Oct 12	9288	785	116.0	91.0
2	Mon, Oct 13	10480	884	145.0	79.0

The unit of observation is the **date**, for each date we have the number of users that arrived to the webpage, that clicked to enroll, who actually enrolled in the free trial and who eventually payed for the service.

Pick your measures

Exercise 3

The easiest way to analyze this data is to stack it into a single dataset where each observation is a day-treatment-arm (so you should end up with two rows per day, one for those who are in the treated groups, and one for those who were in the control group). Note that currently nothing in the data identifies whether a given observation is a treatment group observation or a control group observation, so you'll want to make sure to add a "treatment" indicator variable.

The variables in the data are:

- Pageviews: number of unique users visiting homepage
- Clicks: number of those users clicking "Start Free Trial"
- Enrollments: Number of people enrolling in trial
- Payments: Number of people who eventually pay for the service. Note the `payment` column reports payments for the users who first visited the site on the reported date, not payments occurring on the reported date.

```python
df_control["Treatment"] = 0
df_tmt["Treatment"] = 1

df = pd.concat([df_control, df_tmt])
# df["Date"] = pd.to_datetime(df["Date"], format="%a, %b %d")
print("Shape:", df.shape)
df.head(3)
```

```
Shape: (74, 6)
```

	Date	Pageviews	Clicks	Enrollments	Payments	Treatment
0	Sat, Oct 11	7723	687	134.0	70.0	0
1	Sun, Oct 12	9102	779	147.0	70.0	0
2	Mon, Oct 13	10511	909	167.0	95.0	0

```python
df.groupby("Treatment")["Date"].count()
```

```
Treatment
0    37
1    37
Name: Date, dtype: int64
```

Exercise 4

Given Udacity's goals, what outcome are they hoping will be impacted by their manipulation?

Or, to ask the same question in the language of the Potential Outcomes Framework, what is their Y?

Or to ask the same question in the language of Kohavi, Tang and Xu, what is their *Overall Evaluation Criterion (OEC)*?

(I'm only asking one question, I'm just trying to phrase it using different terminologies we've encountered to help you see how they all fit together)

When you feel like you have your answer, please compute it. Store the average value of the variable in `results` under the key `ex4_avg_oec`. **Please round your answer to 4 decimal places.**

NOTE: You'll probably notice you have two choices to make when it comes to actually computing the OEC.

- You could probably imagine either computing a ratio or a difference of two things — please calculate the difference.
- You may also be unsure whether to normalize by `Clicks`. Normalizing by clicks will help account for variation that comes from day-to-day variation in users, so it's a good thing to do. With infinite data, you'd expect to get the same results without normalizing by `Clicks` (since on average the same share of users are in each arm of the experiment), but for finite data it's a good strategy. Note that this is only ok because users make the choice to click or not *before* they see different versions of the website (it is "pre-treatment").

Just to make sure you're on track, your measure should have an average value of *about* 9%.

```
ex4_avg_oec = np.mean((df.Enrollments - df.Payments) / df.Clicks)

print(f"The proportion of frustrated users is {100 * ex4_avg_oec:.4f}%")

results["ex4_avg_oec"] = np.round(100 * ex4_avg_oec, 2)
```

```
The proportion of frustrated users is 9.4138%
```

Udacity hopes to decrease the number of frustrated users, the ones who enrolled in the platform but didn't become payments. Their *Overall Evaluation Criterion (OEC)* is the amount of frustrated users (Enrollments - Payments) normalized by the number of payments.

Exercise 5

Given Udacity's goals, what outcome are they hoping will *not* be impacted by their manipulation? In other words, what do they want to measure to ensure their treatment doesn't have unintended negative consequences that might be really costly to their operation?

Note that while this isn't how Kohavi, Tang, and Xu use the term "guardrail metrics" — they usually use the term to refer to things we measure to ensure the experiment is working the way it should — some people would also use the term "guardrail metrics" for something that could be impacted even if the experiment is working correctly, but which the organization wants to track to ensure they aren't impacted because they are deemed really important.

Again, please normalize by `Clicks`. Store the average value of this guardrail metric as `ex5_avg_guardrail` and **round your answer to 4 decimal places.**

```
ex5_avg_guardrail = np.mean(df.Payments / df.Clicks)

print(f"The proportion of satisfied users is {100 * ex5_avg_guardrail:.4f}%")

results["ex5_avg_guardrail"] = np.round(100 * ex5_avg_guardrail, 2)
```

```
The proportion of satisfied users is 11.5821%
```

Udacity hopes that the treatment wont affect the satisfied users, ie. the amount of Payments.

It is expected that the pop-up message of the Treatment will discourage people who might not be as commited with Udacity whereas the ones who are really interested in the content will continue with the enrollment. Therefore, the users who aren't as serious will be discouraged from enrolling in the program and as a result, will stay as Clicks not as Enrollments.

Validating The Data

Exercise 6

Whenever you are working with experimental data, the first thing you want to do is verify that users actually were randomly sorted into the two arms of the experiment. In this data, half of users were supposed to be shown the old version of the site and half were supposed to see the new version.

`Pageviews` tells you how many unique users visited the welcome site we are experimenting on. `Pageviews` is what is sometimes called an "invariant" or "guardrail" variable, meaning that it shouldn't vary across treatment arms—after all, people have to visit the site before they get a chance to see the treatment, so there's no way that being assigned to treatment or control should affect the number of pageviews assigned to each group.

"Invariant" variables are also an example of what are known as a "pre-treatment" variable, because pageviews are determined before users are manipulated in any way. That makes it analogous to gender or age in experiments where you have demographic data—a person's age and gender are determined before they experience any manipulations, so the value of any pre-treatment attributes should be the same across the two arms of our experiment. This is what we've previously called "checking for balance," If pre-treatment attributes aren't balanced, then we may worry our attempt to randomly assign people to different groups failed. Kohavi, Tang and Xu call this a "trust-based guardrail metric" because it helps us determine if we should trust our data.

To test the quality of the randomization, calculate the average number of pageviews for the treated group and for the control group. Do they look similar?

```python
df.groupby("Treatment").Pageviews.mean()
```

```
Treatment
0    9339.000000
1    9315.135135
Name: Pageviews, dtype: float64
```

Yes, the average number of Pageviews for both groups are similar, the difference between them is less than 0.5%.

Exercise 7

"Similar" is a tricky concept – obviously, we expect *some* differences across groups since users were *randomly* divided across treatment arms. The question is whether the differences between groups are larger than we'd expect to emerge given our random assignment process. To evaluate this, let's use a `ttest` to test the statistical significance of the differences we see.

Note: Remember that scipy functions don't accept `pandas` objects, so you use a scipy function, you have to pass the numpy vectors underlying your data with the `.values` operator (e.g. `df.my_column.values`).

Does the difference in `pageviews` look statistically significant?

Store the resulting p-value in `ex7_ttest_pvalue` **rounded to four decimal places.**

```python
feat = "Pageviews"

control = df[df.Treatment == 0][feat].values
tmt = df[df.Treatment == 1][feat].values

t_stats, p_value = stats.ttest_ind(control, tmt, equal_var=False)
print(f"p-value for {feat}: {p_value:.4}")

results["ex7_ttest_pvalue"] = np.round(p_value, 4)
```

```
p-value for Pageviews: 0.8877
```

Since the p-value is 0.8877 (> 0.05), then we fail to reject $H_0 : \mu_0 = \mu_1$ meaning that there is no statistically significant difference between the amount of `Pageviews` between the Control and Treatment groups.

Exercise 8

`Pageviews` is not the only "pre-treatment" variable in this data we can use to evaluate balance/use as a guardrail metric. What other measure is pre-treatment? Review the description of the experiment if you're not sure.

`Clicks` is another guardrail metric. As with `Pageviews`, `Clicks` shouldn't be affected by the assignment to the treatment or control group.

Exercise 9

Check if the other pre-treatment variable is also balanced. Store the p-value of your test of difference in `results` under the key `"ex9_ttest_pvalue_clicks"` **rounded to four decimal places.**

```
feat = "Clicks"

control = df[df.Treatment == 0][feat].values
tmt = df[df.Treatment == 1][feat].values

t_stats, p_value = stats.ttest_ind(control, tmt, equal_var=False)
print(f"p-value for {feat}: {p_value:.4}")

results["ex9_ttest_pvalue_clicks"] = np.round(p_value, 4)
```

```
p-value for Clicks: 0.9264
```

Since the p-value is 0.9264 (> 0.05), then we fail to reject $H_0 : \mu_0 = \mu_1$ meaning that there is no statistically significant difference between the amount of `Clicks` between the Control and Treatment groups.

Estimating the Effect of Experiment

Exercise 10

Now that we've validated our randomization, our next task is to estimate our treatment effect. First, though, there's an issue with your data you've been able to largely ignore until now, but which you should get a grip on before estimating your treatment effect — can you tell what it is and what you should do about it?

Store the number of observations in your data *after* you've addressed this in `ex10_num_obs` (this is mostly meant as a way to sanity check your answer with autograder).

```
print(
    f"{df.shape[0]} original observations which correspond to {df.Date.nunique()} unique dat
)

data = df[~df.Enrollments.isna()].copy()

ex10_num_obs = data.shape[0]
```

```python
print(
    f"{ex10_num_obs} observations without missing values which correspond to {data.Date.nuniq
)

results["ex10_num_obs"] = ex10_num_obs
```

```
74 original observations which correspond to 37 unique dates
46 observations without missing values which correspond to 23 unique dates
```

The issue is that originally we had 74 observations corresponding to 37 unique dates; however, 14 of such dates missing values.

Exercise 11

Now that we've established we have good balance (meaning we think randomization was likely successful), we can evaluate the effects of the experiment. Test whether the OEC and the metric you *don't* want affected have different average values in the control group and treatment group.

Because we've randomized, this is a consistent estimate of the Average Treatment Effect of Udacity's website change.

Calculate the difference in means in your OEC and guardrail metrics using a simple t-test. Store the resulting effect estimates in `ex11_oec_ate` and `ex11_guard_ate` and p-values in `ex11_oec_pvalue` and `ex11_guard_pvalue`. **Please round all answers to 4 decimal places.** Report your ATE in *percentage points*, where 1 denotes 1 percentage point.

```python
data["OEC"] = (data.Enrollments - data.Payments) / data.Clicks
data["GUARDRAIL"] = data.Payments / data.Clicks
```

```python
feat = "OEC"

control = data[data.Treatment == 0][feat]
tmt = data[data.Treatment == 1][feat]

t_stats, p_value = stats.ttest_ind(control, tmt, equal_var=False)

ex11_oec_pvalue = np.round(p_value, 4)
print(f"p-value for {feat}: {p_value:.4}")

ex11_oec_ate = np.mean(tmt) - np.mean(control)
```

```
ex11_oec_ate = np.round(100 * ex11_oec_ate, 4)
print(f"Difference in means for {feat}: {ex11_oec_ate} percentage points")

results["ex11_oec_ate"] = np.round(ex11_oec_ate, 4)
results["ex11_oec_pvalue"] = np.round(ex11_oec_pvalue, 4)
```

```
p-value for OEC: 0.132
Difference in means for OEC: -1.5888%
```

```
feat = "GUARDRAIL"
```

```
p-value for Guardrail (Payments / Clicks): nan%
```

Exercise 12

Do you feel that Udacity achieved their goal? Did their intervention cause them any problems? If they asked you "What would happen if we rolled this out to everyone?" what would you say?

As you answer this question, a small additional question: up until this point you've (presumably) been reporting the default p-values from the tools you are using. These, as you may recall from stats 101, are two-tailed p-values. Do those seem appropriate for your OEC?

Exercise 13

One of the magic things about experiments is that all you have to do is compare averages to get an average treatment effect. However, you *can* do other things to try and increase the statistical power of your experiments, like add controls in a linear regression model.

As you likely know, a bivariate regression is exactly equivalent to a t-test, so let's start by re-estimating the effect of treatment on your OEC using a linear regression. Can you replicate the results from your t-test? They shouldn't just be close—they should be numerically equivalent (i.e. exactly the same to the limits of floating point number precision).

Exercise 14

Now add indicator variables for the date of each observation. Do the standard errors on your `treatment` variable change? If so, in what direction?

Store your new standard error in `ex14_se_treatment`. Round your answer to 4 decimal places.

You should have found that your standard errors decreased by about 30%—this is why, although just comparing means *works*, if you have additional variables adding them to your analysis can be helpful (all the usual rules for model specification apply — for example, you still want to be careful about overfitting, which one could argue is maybe part of what's happening here).

In many other cases, the effect of adding controls is likely to be larger — the date indicators we added to our data are perfectly balanced between treatment and control, so we aren't adding a lot of data to the model by adding them as variables. They're accounting for some day-to-day variation (presumably in the types of people coming to the site), but they aren't controlling for any residual baseline differences the way a control like "gender" or "age" might (since those kind of individual-level attributes will never be perfectly balanced across treatment and control).

Exercise 15

Does this result have any impact on the recommendations you would offer Udacity?